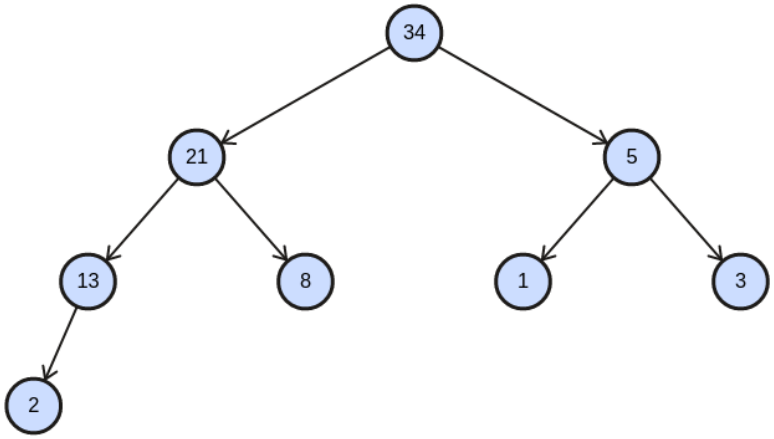
Aluno: Luís Eduardo Bertelli
ExeHeap-1
O seguinte algoritmo foi utilizado para resolução das questões:

heapify.c

```c
#include<stdlib.h>


int left(int i) {return (2* i + 1);}
int right(int i) {return (2*i+2);}

void heapify(int *a, int n, int i){
    int e, d , max, aux;
    e = left(i);
    d=right(i);
    if(e < n && a[e] > a[i])
        max = e;
    else
        max = i;
    if(d < n && a[d] > a[max])
        max = d;
    if(max != i){
        aux = a[i];
        a[i] = a[max];
        a[max] = aux;
        heapify(a, n, max);
    }
}

void buildHeap(int *a, int n){
    int i;
    for(i = (n-1)/2; i>=0; i--){
        heapify(a, n, i);
    }

}
int main(){
    int i;
    int n = 8;
    int a[8] ={2, 5, 8, 13, 21, 1, 3, 34};

    buildHeap(a, n);
    printf("[ ");
    for (i = 0; i < n; i++){
        printf("%d ", a[i]);
    }
    printf("]\n");
    return 0;
}
```

Questão a)
Montagem da Heap Máxima:



Após a execução da buildHeap, a heap será graficamente representada pela imagem acima, num formato mais textual, o arranjo ficará int a = [34, 21, 8, 13, 2, 1, 3, 5].
Com uma execução do código acima, isso se confirma em:

```
luis2535@luis2535-IdeaPad-3-15ALC6:~/UDESC/CAL/HeapEXE$ ./heapify
[ 34 21 8 13 2 1 3 5 ]
```

b)
No pior caso da função heapify, o cálculo a seguir representa a complexidade de tempo para o pior caso:

$$\sum_{i=0}^{n} (n-1)\,2^{i}$$

$$= \sum_{i=0}^{n} (2^{i} * n - 2^{i})$$

$$= \sum_{i=0}^{n} 2^{i} * n + \sum_{i=0}^{n} 2^{i}$$

$$= n \sum_{i=0}^{n} 2^{i} + \sum_{i=0}^{n} 2^{i}$$

$$= n(2^{n+1} - 1) - (n2^{n+1} - 2^{n+1} + 2)$$

$= n2^{n+1} - n - n2^{n+1} + 2^{n+1} - 2$

$= -n + 2^{n+1} - 2$

$= -n + 2^{n+1} - 2$

$= -n + 2 * 2^{n} - 2$

$= n = log2\ N$

$= -log2\ N + 2n - 2$

$O(n)$

Dessa maneira, foi calculado que a complexidade de tempo da função é O(n).